Michael W. Evans

Mr. Michael W. Evans is a recognized expert in software engineering, management and project control, system and software methods, and the engineering process. He is President and founder of Integrated Computer Engineering, Inc., a software engineering consulting firm specializing in software engineering methods and processes. He has written four books published through John Wiley & Sons. Recent efforts have been in support of the DoD Software Acquisition Best Practices Initiative, including the Software Program Managers Network, a forum for software development professionals to share lessons learned and a distribution vehicle for the Best Practices Initiative. He is a member of the Airlie Software Council, a key element of the Initiative.

Michael W Evans
Integrated Computer Engineering Inc
Central Ave
Campbell CA 95008

Voice: 408-378-4700
Fax: 408-378-5395 142 N
E-mail: candca@aol.com

Title: CenterZone Management™: The Relationship Between Risk Management and Configuration Management in a Software Project
Presenters: Michael W. Evans and Shawn T. O'Rourke
Track: 10
Day: Tuesday, April 29
Keywords: CenterZone Management™, configuration management, risk management, chaos, process improvement
Abstract: Software managers must apply two very different models simultaneously if their projects are to succeed. On the one hand, they must make certain the project environment retains rigor and discipline to ensure the product meets predefined requirements. On the other hand, they must allow software engineers sufficient freedom to operate in an unstructured creative environment that enhances productivity. Successful managers establish a balance between chaos and control. They manage in the "CenterZone," moving back and forth between chaos (freedom) and control (discipline) to satisfy project needs.

CenterZone Management™:

The Relationship Between Risk Management and Configuration Management in a Software Project

Much has been written over the last 25 years describing the "Software Crisis." The crisis, first identified at the 1968 NATO conference in Garmisch, Germany, has caused almost continuous government, industry and academic concern about the future of the software industry.

In today's Department of Defense (DoD) acquisition environment, software remains as much of a concern as it was in 1968. Software problems continue to preclude the development or augmentation of critical weapons and automated information systems.

There is no single estimate of the number of dollars invested in software process improvements, but the figure is known to amount to many billions. Despite this vast expenditure on developing improvements, software problems continue to plague the defense industry, and industry in general.

A recent report by the General Accounting Office (GAO) states that, although the federal government spends billions of dollars annually on information technology, it is unclear what the government is getting in return for its money. Better facts are needed about the government's information technology movements. Although information technology can boost organizational performance, the risks of failure are ever present and must be effectively managed.[1] To be successful, software-intensive projects must be managed with a balance between a rigorous software development process and innovative management techniques.

Software is a major cost, schedule, and performance driver of virtually all DoD weapons, command-and-control and information systems. Of the estimated $42 billion that the DoD spends annually on the development and maintenance of its computer systems, just $7 billion—one sixth—is spent on hardware. As of July 1995, the DoD had more than $256 billion under contract for software-intensive

---

[1] GAO/T-AIMD-97-38.

systems. When software problems delay the fielding of a major system, the costs can become enormous.

Software problems are not unique to defense systems. Many large software projects now underway will experience significant defects and problems, and may even be canceled after very substantial investments. Despite significant national expenditure in the area of software process improvement, have we made corresponding improvements in the bottom-line metrics of cost, schedule, quality and user satisfaction?

Cost and Schedule Performance

Problems in software development currently have a significant impact on industry in general. A 1995 study found that only 16 percent - one sixth - of software projects are expected to finish on time and on budget, with only 9 percent of software projects in larger companies being completed on time and within budget. Less than half—just 42 percent—of the originally proposed features and functions will be present in projects completed by the largest U.S. organizations. Fifty-three percent of projects will cost 190 percent—almost double—of their original estimates. Some 31 percent of software projects are canceled before completion.[2] An estimated $81 billion was spent on canceled software projects by U.S. companies and government agencies in 1995. As large-scale software systems have become commonplace, the ability to manage these systems effectively has not kept pace. In the U.S., development costs continue to rise while productivity declines. Virtually all nations with highly qualified software practitioners have development costs between $125 and $250 per function point, except Japan at $1,600 and the U.S. at $1,000.[3]

---

[2] "Chaos," *Open Computing*, March 1995.

[3] Capers Jones, *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 1991, pp. 142, 145, 149, 167.

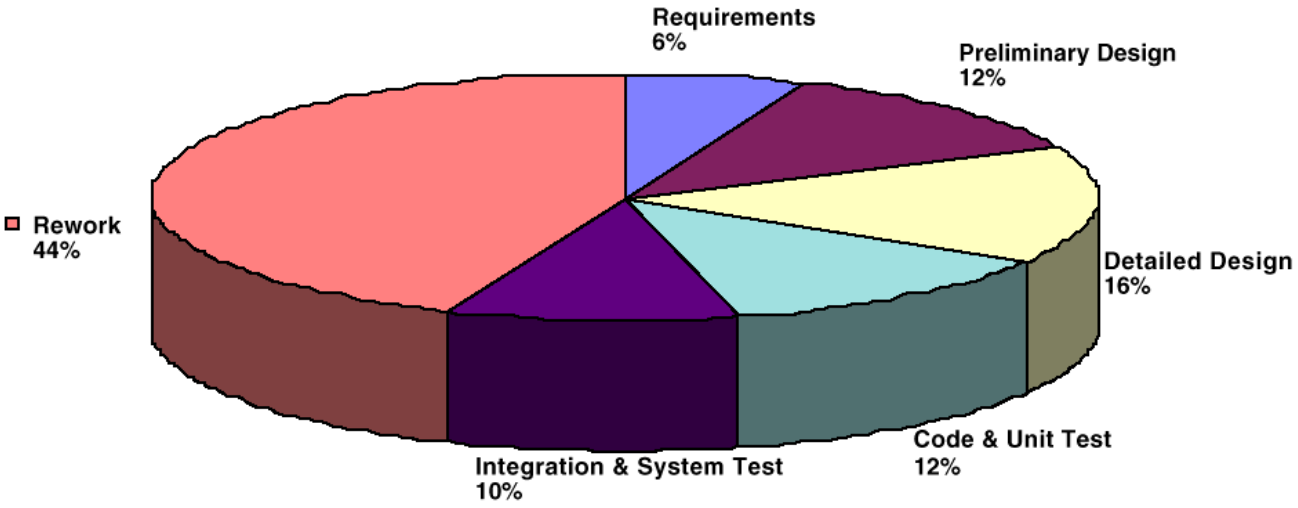

Figure 1. Rework cost required for defect removal.

System development costs have been increasing exponentially with project size. Management of system development has been found to be a key determinant of cost. If U.S. software managers do not improve how they manage their large systems, they could see the nation's software business move overseas.

Software Quality and User Satisfaction

The number of defects occurring in software delivered to the field is not a linear function of size—defects increase exponentially. In a study by Capers Jones, he determined that defect potential is directly related to the function point total of an application raised to the 1.2 power.[4] As software projects become larger, defects become a much more significant problem. As illustrated in Figure 1, rework (the cost of finding and fixing a defect) approaches half the development cost in large software projects. Thus efforts to identify and eliminate defects, and prevent their introduction, must be a major component of any strategy to reduce the cost and risk of software development.

The GAO has identified poor software management and a lack of project oversight as critical and common problem areas. An effective assessment process, to give early warnings of potential problems before their full impact is felt, is essential to addressing the GAO's concerns.

Improving Quality

As illustrated in Figure 2, consistently effective management practices, focused on defect reduction, can significantly reduce the number of defects and their root causes. These management practices enable defect identification, monitoring, reporting and closure. Defects can be minimized through process and product standards enforced through integrated quality management practices.

All managers want their programs to succeed, yet they face a myriad of challenges, both externally and internally generated, that can impact overall success and force decision makers into choices between undesirable alternatives. To achieve success, managers must cope with schedule, budget, resource and technical restraints. In addition, varying program requirements complicate management plans. Unanticipated change in any of the aforementioned factors can impact the others, and force trade-offs. The trade-off between a delivery date and a required capability is not an easy management decision. In some cases, funding may expire and necessitate acceptance of a lesser system to avoid the possibility of cancellation.

---

[4] *Ibid.*

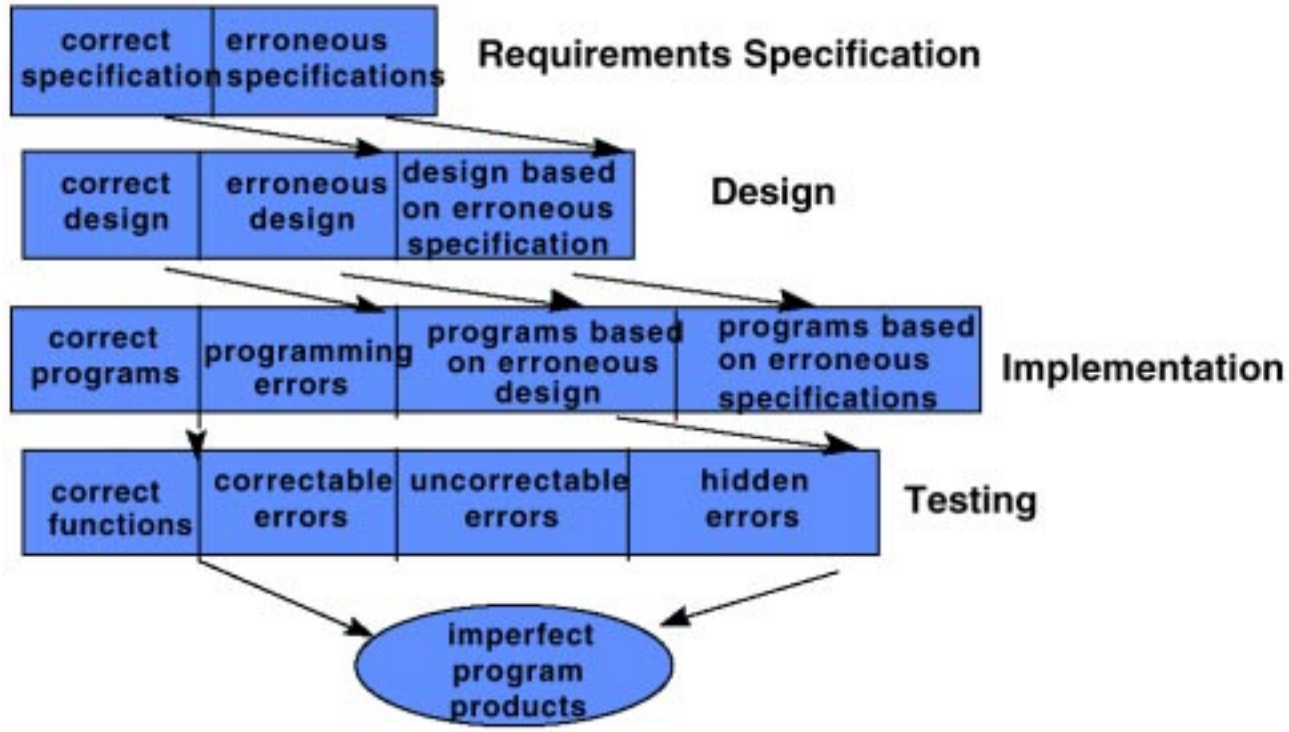

Figure 2. Some causes of software defects.

In all cases, program management is in pursuit of success, yet when faced with the reality of the acquisition environment, the final outcome may not be the product and capabilities originally envisioned. As illustrated in Figure 1, if we don't manage towards defect removal we will continue to experience rework costs on software projects which exceed development costs in the late stages.

Yet, despite problems, differences in process, and obvious inefficiencies and confusion surrounding software standards, processes, methods and tools, we continue to develop and deploy remarkable products. Spurred by the DoD, the acquisition environment has changed and continues to change. An example of this is the decreased reliance on military standards and the use of commercial standards and commercial off-the-shelf (COTS) products. The restructuring of software acquisition management and use of Integrated Product and Process Development and Integrated Product Teams continue to yield results. There is new direction in performance-based specifications, and the oversight process is changing from adversity to a streamlined teaming approach. The acquisition environment has improved and continues to improve. The systems fielded attest to the skill and creativity of our people.

Common Characteristics of Complex Projects

Complex systems—and **a software development project is a complex system**— exhibit certain common characteristics that could lead to unexpected development outcomes if the systems are not properly managed. These characteristics cannot be explained by analyzing individual aspects of the software project, but can only be addressed by understanding and addressing the relationships between them. Software planning which breaks the project into its smallest parts, and then details the "how" of each part, is useful in understanding the essential project components that must be present to enable success and potential cost, resources and organizational considerations necessary to implement these in the project environment. This low-level planning will not, however, address the risks associated with implementing specific disciplines in the project. Understanding user-, system- and software-product interfaces and the relationships, interactions and risks resulting from process interactions is critical if risks are to be projected, managed and resolved.

In complex systems, the interesting and relevant project behavior that leads to risk occurrence arises from the spontaneous interaction of the individual project activities and organizational components. These interactions can't be planned or directed; they just happen. They result from project reactions to random external and internal events and occurrences. The project structure and environment must be "self-organizing"; it must be capable of being dramatically restructured quickly without destroying the flow of work or incurring even more risk. If the project is at either pole—either overly chaotic or overmanaged—reacting to risk is often impossible. In software terms, **chaos** defines an environment influenced by internal or external events that adversely affect a desired outcome.

Software project management needs to be flexible, exhibiting self-organizing behaviors that respond easily to changing program needs, requirements, realities and constraints, without losing essential discipline and control. They must recognize the maximum software project risks. These risks, whose occurrence will destroy a project or cause a product not to be fielded, occur at the juncture of two activities or organizations, or the interface points between two products. They are harder to anticipate; by the time they are recognized the impact has been experienced, and correcting or resolving problems resulting from the occurrence often requires the involvement of more than one organization.

Let's look at the implications that this has on software project success. On one hand you have the government contracting and acquisition environment. In this environment, specific requirements and basic technical parameters are defined, negotiated between acquirer and developer, and frozen as the basis for the system acquisition and, ultimately, the basis for the software requirements. Inflexibility of requirements, while essential in establishing agreed-to relationships between the user, the acquirer and the developer, move the software project away from the point of balance towards the rigid, ponderous state indicative of too many DoD software acquisition projects. Other forces tend to drive the DoD software acquisition project further in this direction. The contracting environment fostered by the DoD fixes specific agreements between acquirer and developer and, by intent, makes it difficult, often impossible, to change them. Program management, engineering, assurance and reporting disciplines are often ponderous, moving at glacial speed to resolve immediate, short-term software problems and issues. Because of the number of organizations that may have to concur to proposed program changes, and the complexity of the analysis required to allow this concurrence, critical changes to technical parameters may take months, sometimes years, to be resolved. The rigidity of this environment, coupled with the extensive delays associated with contract change, forces software projects into a fixed structure far from the ideal "edge of chaos" environment critical to project success.

The other side of the equation is the pressure that customers and program and software-project management place on the software engineering staff to be infinitely responsive to requested modifications to the project structure and product characteristics. This responsiveness is consistent with the stereotype of software engineering. The view is that, as long as software engineers are provided a free and unfettered environment and allowed to pursue technological innovation, significant change to the software is possible at low risk. This, coupled with the need for software to be modified frequently to resolve problems, the lack of funds or resources to establish and maintain a disciplined software project environment, and the inconsistency between a rigorous and tightly controlled software environment and the culture of the software project, tends to push the project away from the edge of chaos and into the spiral of anarchy.

The Department of Defense has taken dramatic steps to attempt to deal with the issues which impede acquisition of software. General changes in the acquisition environment are being enabled

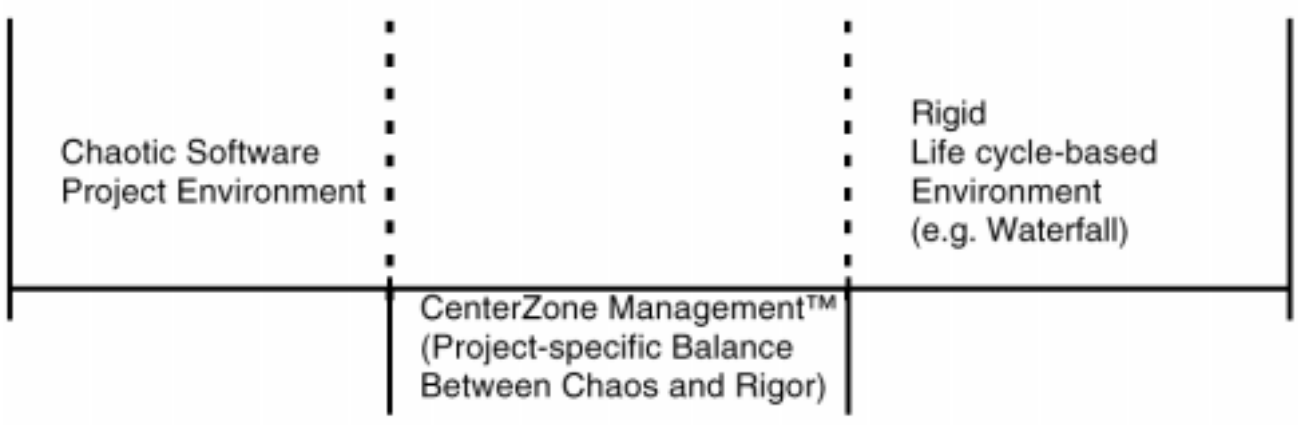

Figure 3. CenterZone Management™.

through acquisition reform—revisions to the Federal Acquisition Regulations (FAR) and rewrites of the 5000 and 8000 series; Acquisition Management Policies and Procedures. Dramatic changes such as the Perry Memo, which replaces cumbersome DoD specs and standards with commercial practices and MIL-STD 498, move the software standards away from the old waterfall life cycle model and closer to an activity-based software standard more representative of how software is developed, delivered and supported. These initiatives do not, however, address the project need to dynamically establish and maintain a meaningful balance between chaos and the rigid, structured environment incapable of rapid and responsive change. The Best Practices Initiative has addressed this requirement.

This initiative has identified 9 principal and 43 supporting practices and a variety of tools that a manager can use to monitor project effectiveness, risk and potential for success. The practices and tools come from successful organizations which have used them to manage the chaos related to software engineering and acquisition. These practices generally address interfaces and relationships between specific project activities, the threads tying the project together. As such, they enable the project to maintain the balance between chaos and over-control. They allow the manager to retain the “edge of chaos” balance, providing early warning when the project is headed out of control. This allows the manager to step in to manage the chaos, instead of having the chaos manage the project. This is known as *CenterZone Management*.™

CenterZone Management™

The authors have made an interesting observation while conducting assessments of successful and unsuccessful software projects for the past 10 years. Project managers who follow a rigorous, fixed process that cannot adapt to the dynamic management, technical, and changing operational needs of their customers have the same or greater risk of failure than managers who let projects drift into chaos. Being flexible is as critical to the success of a software project as technical excellence.

As illustrated in Figure 3, software projects must apply two very different models simultaneously if they are to succeed. On the one hand, the project must rigorously maintain baselines and agreements between users, related software and systems, and customers. By necessity the software project environment must retain rigor and discipline, building a product that meets predefined requirements. On the other end of the scale, software engineers must be provided sufficient freedom to implement software in a free and unstructured environment, applying heuristic methods with skill and technical understanding. If talented engineers are overmanaged, this will adversely affect the culture of a project and limit or destroy productivity. Successful managers establish a balance between control and chaos, providing sufficient discipline to assure product acceptability while drifting into controlled chaos, when necessary,

to enable the development of software. Effective managers manage in this "CenterZone," moving back and forth between chaos and control when necessary to satisfy project needs.

What are the effects of CenterZone Management™? Projects that maintain a balance between chaos and overcontrol lower the risk of problems in the bottom-line metrics of cost, schedule, quality and user satisfaction.

Dynamic Process Adaptation

Projects that successfully manage in the CenterZone employ the concept of "Dynamic Process Adaptation." In a software project environment, the ability to adapt is an essential characteristic of a successful project. Successful software project managers balance the need for order and the imperative for change. Their projects tend to locate themselves at "the edge of chaos." The edge of chaos is where there is enough innovation to keep a project functioning while retaining sufficient order and stability to keep it from collapsing into anarchy. The successful software manager walks the narrow line between order and discipline and pure chaos, finding the balance point.

Figure 4. (missing) The balancing of chaos and rigidity in software development process.

This balance point must be carefully maintained if the project is to succeed. If the organization drifts too far into chaos, it risks falling into disorder. If this state is reached, the project becomes incapable of accomplishing preplanned tasks, and even the simplest accomplishments become impossible. If the project moves too far in the other direction away from the balance point, it becomes rigid, frozen, and impossible to adapt to the reality that it faces. Either of the two conditions leads to failure and may be a major cause of many of the software project failures plaguing the industry. Too much change can be equally as destructive as too little. Only at the edge of chaos can software projects succeed.

Figure 5. (missing) The balancing of configuration management information and risk information in software development process.

The two basic disciplines that enable software management in the CenterZone are Configuration Management and Risk Management.

Configuration Management: An Essential for CenterZone Management™

As illustrated in Figure 6, Configuration Management (CM) allows the project to manage information that is either shared by organizations within the project or which has been approved through a quality gate (an inspection, walkthrough, audit or review) for use in the project. CM is a discipline applying technical and administrative direction and surveillance to:

1. Identify and document the functional and physical characteristics of a configuration item.
2. Control changes to those characteristics.
3. Record and report change-processing and -implementation status.

Figure 6. (missing) CM provides dynamic control based on information approval.

When managing in the CenterZone, the least amount of CM control is on information not shared or not yet approved. At this level of control, the software engineer manages content and change without the need for approval or review. Any use by other individuals or organizations is at risk. While in this state, the activity being conducted most closely matches the chaotic project state. At some point, either the engineer or his or her task leader schedules a quality gate to evaluate a completed work product(s). Three actions may be taken as a result of the gate evaluation:

1. The product is approved without change.

Test Requirement Specifications	Software Design Specification	Software Design Specifications	Software Design Specifications	Subsystem Functional Specifications Test Readiness	System Specifications	System Requirements Specification	Operational Interfaces	Operational Performance
Test Review	Unit Test Walkthrough	Unit Test Walkthrough	Build Test Walkthrough	Functional Configuration Audit	Program Acceptance Review	System Acceptance Review		Operational Acceptance
Test Level	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
	Module Test	Unit Test	Software Subsystem Integration	Software Subsystem Functional	System Integration	System Functional	Operation Integration	Operation Functional
Test Objective	Module Implementation	Unit Design/ Functional Allocations	Software Subsystem Integration	Software Subsystem Functional	Software Integ. A s'ware/h'ware B s'ware/h'ware	System Performance	Interfaces	Support
Test Planning		Module Documentation	Build Test Plan	Software Test Plan	System Test Plan	System Test	Operators'/ Users' Manuals	Operators'/ Users' Manuals
Test Specification		Module Documentation	Build Test Folder Procedures Scenarios	Software Test Procedures Scenarios Data	Test Folders Procedures Scenarios Data	Test Procedures Scenarios Data	Test Procedures	Manuals'/ Operators' Goals

Figure 7. Test flow versus project control.

2. The product is approved, although discrepancies are opened as problem reports.
3. The product fails the gate and the product is returned for rework.

Products that pass the gate are placed under internal control, first by the CM librarian staging the product release in a CM-owned library partition, and then by moving the product into the controlled project partition. Ownership of the information has moved from the engineer to the software manager. The project, at least for the information transferred, has moved from potential chaos to the CenterZone where sufficient controls are in place to ensure that:

1. Multiple individuals using the same information use a common, approved version, minimizing rework and potential for defects creeping back in after a quality gate completes.
2. Unauthorized changes are not made to products which would impact the users of the product. Changes are not made until the impact is understood and agreed to.
3. Engineering changes are tasked based on need, the process is managed, and the products are re-evaluated before the modification is incorporated into a release.

Information in the controlled project partition cannot be modified unless the change is authorized by the Engineering Review Board (ERB). Examples of these changes are:

1. Changes to the preliminary specification(s) prior to the final product baseline.
2. Any change to products released during development testing or to a functional area during development testing, unless the change affects the documentation representing the functional or allocated baselines.
3. A change that does not affect a baseline but does impact internal project information, capabilities or support.

The ERB maintains management control over the form, content and structure of shared information. The ERB evaluates all proposed changes to software and data not yet accepted by the customer, which have been moved into the project control partition of the software library. The board reviews and evaluates all System Problem Reports (SPRs), authorizes the appropriate corrective action to be taken, tracks status of all changes in progress, and closes the SPRs after the changes have been made. The board is technical, screening all reported problems for impact and determining whether the correction is in the

scope of the project.

The ERB is a technical, not management, board. Its primary purpose is to ensure that changes to information not yet approved by the customer are in the scope of the project, are technically feasible and required, and that all impacts are known. If the proposed change is “out of scope,” the SPR is sent to the Change Control Board for action. Engineers assigned work by the ERB check out the data to change from the project control partition of the library. The manager has reestablished project control, essential to avoiding chaotic change by the engineering organization.

Information that has been approved for testing through the Test Readiness Review (TRR) quality gate is moved from the project control partition to the testing partition. This area of the library is owned by the test manager and contains all test-specific information, object builds of the software under test, all test tools and data, all test scenarios, and the Version Description Documentation for all software under test. No changes to this library partition can be made unless authorized by the ERB and accepted by the test manager.

As illustrated in Figure 7, testing is a seamless structure comprised of eight distinct test levels. Level 1 and Level 2 testing are the responsibility of the software engineer developing or modifying the software. Except for defining standards and practice requirements for this testing, the project manager pretty much relies on the engineer to adequately test the software. The process is, however, checked at a quality gate. When the gate successfully completes, the software is moved into the project control partition. Integration (Level 3 tests) is performed by the test manager using software configurations built from the project control partition into the test partition. Management control has moved into the CenterZone, providing essential controls without overburdening the process with management overhead.

Any problems identified are documented on SPRs and submitted to the ERB for disposition. This plan-execute-expose-report-correct process is followed until the software completes Level 6 test. To minimize project risk, all software builds from Level 3 through Level 6 should use software from the project control partition and be documented through a Version Description Document.

Software that has successfully completed Level 6 testing, or documentation that has been internally reviewed and is approved for release to the customer through an internal audit, is moved into the pre-release partition of the library. This partition is owned by the program manager. By design, change is difficult due to the number of organizations impacted by problems, and the impact of changes. When information moves into this partition, the project has moved out of the CenterZone and into a process of excessive control. As reviews are completed and products accepted by the customer, information is placed in formal baselines owned by the customer and residing in the customer partition of the library. Information included in the functional, allocated or product baselines in the customer partition can only be changed through a Change Control Board (CCB)-approved Engineering Change Proposal (ECP).

The CCB, unlike the ERB, addresses management issues rather than technical issues. Problems that are observed and documented on an SPR are initially submitted to the ERB. The ERB screens them to determine impact, scope and products affected. If the impact is considered major, if the SPR correction is outside the scope of the project charter, or if the SPR affects information previously approved by the customer and included in a jointly managed baseline, the SPR is converted into an ECP and dealt with at a CCB meeting.

This CCB process is high risk for the project for several reasons:

1. The CCB is slow since approval of ECPs requires coordination from many affected groups and contractual actions.
2. Since it is a management board, understanding of the technical implications as the basis for decisions may be difficult.
3. Without initial impacts being provided by the ERB, screening decisions may be based on a programmatic, rather than a complete, understanding of the technical difficulty or cost of the proposed change.

When managing change affecting customer-approved information or change that passes through the CCB process without ERB screening, the project moves from the CenterZone into a high-risk area.

Risk Management: The Other Essential Discipline

How does risk management play in this process? It is difficult to comprehend why software managers can't anticipate the catastrophes they experience, and take steps to avoid potential impacts. The reason may be that, at the basic program level when scheduling the occurrence of events, projecting and managing cost and resource expenditures and projecting progress are not areas that can easily be controlled. Events regulate progress, and many of these events are outside software project management control.

Risk management is a discipline which enables a software manager (and anyone else in the program) to anticipate problems before they occur and thus minimize the impact on the project. In CenterZone Management™, several risk management-related definitions are important:

1. **Risk:** A problem that has not yet occurred.
2. **Problem:** A negative impact resulting from occurrence of a risk.
3. **Risk Transition:** What happens to cause a risk to become a problem.
4. **Risk Management:** What is done to assess and control risks prior to transition.

When the project is in the CenterZone, information is in the controlled partition of the library and is either under test or being evaluated through quality control, quality assurance or other form of analysis. While there may be reported problems against releases under project control, or of individual components comprising a release, the problems are known, the risks are identified and are usually being tracked. Metrics are identified and monitored quantitatively to determine when the likelihood of occurrence of specific risks passes a predetermined threshold.

During each software development stage—i.e. requirements definition, design, coding, and test—the controls for the associated products of these stages rest with the responsible engineer. At each stage the engineer is responsible for achieving and meeting predetermined quality gates in order to transition the product to the next development stage. Only when the product transitions to integration testing does the individual engineer lose direct control of the project. The integration stage brings all engineers together to move the product to the production stage through team resolution of issues. Management attempts to overmanage or overcontrol this process will interfere with the creative process, and will certainly increase risk and impact productivity.

When information and process are under the engineers' personal control, the risks cannot be as well managed, and it is harder to define and collect metric data. Engineers will interpret when requirements are ambiguous, they will be optimistic when optimism is unwarranted, they will not always follow

standardized processes or develop products which match project standards. Also, when problems are assigned to engineers for analysis or correction through an ERB or CCB action, and information is checked out for change, the project risk management process looks like the development model.

Risk Management Options

Every program manager has to deal with a multitude of unidentified risks that are posed and waiting to interject themselves in the acquisition process. These risks can be as simple as an alarm-requiring- reset risk, to a risk that would cause major program-funding reduction. A variety of risks in some form or other will undoubtedly surface. The question is how the program manager controls risks to ensure the program successfully achieves cost, schedule and performance requirements. The proactive manager can apply four techniques to control acquisition risks: 1) **Avoid**, 2) **Contain**, 3) **Mitigate,** 4) **Evade.** All but evasion need cultural insertion to affect an institution focus on risk management.

For today's program manager, a critical decision point in the acquisition process is how to balance his attention on all aspects of the acquisition process while avoiding the inherent risks facing the program. Risk management issues are not only the concern of the program manager but of his prime contractor as well. How well the prime contractor and his supporting subcontractors interact to offer their viewpoint of program risks to the program office is critical. The CenterZone is achieved with a complete team.

The first approach—**Risk Avoidance—**is a true risk. Here the program manager purposely decides a plan of action where he willingly positions the program where it may be susceptible to unexpected risks, yet he takes no action to prevent them. The program manager is willing to absorb the risks in terms of schedule, cost and performance, if they surface. In some cases the program manager believes the potential occurrence cost is far less costly than putting in a plan to mitigate the potential risk area. This decision could be costly in itself as risk/benefit analysis has been conducted with incomplete program development support data. With at least 73 percent of software developers at SEI CMM maturity level 1, the reliability of supplied data could be questionable. Compounding this situation is a program office which takes no action to avoid potential bad effects. Sometimes the program manager is faced with a situation where there are no risk options available. A program manager working on an already tightly compressed and financially constrained program is ill-advised to disregard understanding his risk options and recoveries.

The program team that recognizes risks are inevitable will develop a strategy focusing on managing risks for the betterment of the program acquisition. The best approach is to focus on **Risk Containment**. An active risk management program maintains risk containment as a key element of successful acquisition. The challenge that now faces the program team is determining the probability of each risk occurring. Through an active risk identification process, the team calculates the probability of the risk occurring, the cost of the occurrence if it does happen, and multiplies these factors to determine a risk reserve. For example, if late delivery will cost $100,000 in penalties, and has a 30 percent chance of occurrence, the risk reserve is $30,000. Cumulatively these risk reserves reflect the total budgetary reserve required to be set aside to manage a risk's occurrence. Unfortunately, no program can establish a risk reserve of sufficient quantity to cover all risks.

Can a program team successfully manage a program with a limited amount of risk reserve? The answer is no, as risks without sufficient reserves often can't be mitigated successfully. Therefore it is imperative that a program team set up a reasonable reserve which should include time, dollars, resources,

or any other limited item to ensure the risk event or item remains off the critical path. The enterprising program team will maintain a relative position in the CenterZone to balance out potential risks equally.

The successful team can avoid expending risk reserves if each risk is examined and a **Risk Mitigation** plan is developed, monitored and implemented when necessary to avoid the risk. Each risk should be re-examined periodically to ensure no changes have occurred in either the probability or cost of the occurrence. Mitigating a risk early in the development process is less costly than when it has become an established problem. The risk that becomes an established problem will consume more reserves than previously. So what steps does a program team have to implement to be positioned successfully in the CenterZone? To begin with, determine an alternate strategy or process in advance of a risk transition to a problem in order to minimize the impact of the risk should it materialize. For example, set up a second source for hardware in the event a hardware delivery is late. To avoid unacceptable impacts, non-negligible risks must have mitigation strategy and reserve requirements identified. The monitoring approach must be institutionalized throughout the program team. This monitoring requires unambiguous, quantitative triggers to initiate the mitigation option. Again, a successful team is identified as operating in an open environment that is responsive to changes. This environment requires balancing within the CenterZone region. No one action can mitigate a risk, but a balancing among actions is required.

There are situations in every program where **Risk Evasion** is more the norm. The environment tends to be reactive to events instead of proactive. In some situations a known risk will eventually transition into a problem; in others, the expected probability is so low that the risk is ignored. Each program team has to determine what to do about each risk. In some cases the risk has little opportunity to impact the development's critical path, or the cost of occurrence is so low and is absorbable. In these cases the program team basically ignores the potential of the risk becoming a problem. The program presses on and hopes for the best. As you can see, evasion is cheap, requires no action and removes any option for mitigating the risk. This is the option observed in most DoD programs. For commercial programs supporting DoD programs, risk evasion is common unless the program sponsor provides adequate funding for a more effective risk management option. The viewpoint here is that risk evasion is common when no alternative risk management process exists.

The best approach to identifying and managing risk is through education and developing a **Risk Management Process**. A successful risk management process starts with establishing a risk management program sponsored by the highest level of management. The key process steps are: 1) designating a risk officer, 2) risk identification, 3) risk analyses and prioritization, 4) decriminalizing risk, 5) risk reporting, 6) establishing a risk reserve, 7) establishing a continuous, visible risk management program. Once the program is established a **metrics-based risk management** approach should be implemented. This applies a quantitative approach to assessing and controlling risk. A proactive and responsive program team that implements a program-wide risk management program will certainly be found in the CenterZone.

Summary

Today's environment of budgetary constraints, rapidly retiring legacy systems and staff downsizing are symptoms of the chaos facing the software engineering community, the acquisition force and ultimately the customer. In a proactive management environment, chaos fuels the need for system engineering improvements. System improvements that rely on process improvement alone do not strengthen the system engineering environment. The reactive management environment looks at process improvement as the silver bullet. The environment is too dynamic to leave to one initiative or management approach.

The key factor in managing the system engineering environment is to achieve the necessary balance in the CenterZone. As we move forward into the 21st Century, achieving the critical balance in the CenterZone is the key to achieving the maximum return on investment for every investment dollar. Are you prepared to make the shift in thinking, managing and reacting in the CenterZone?